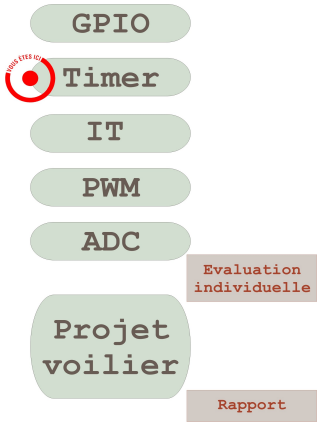
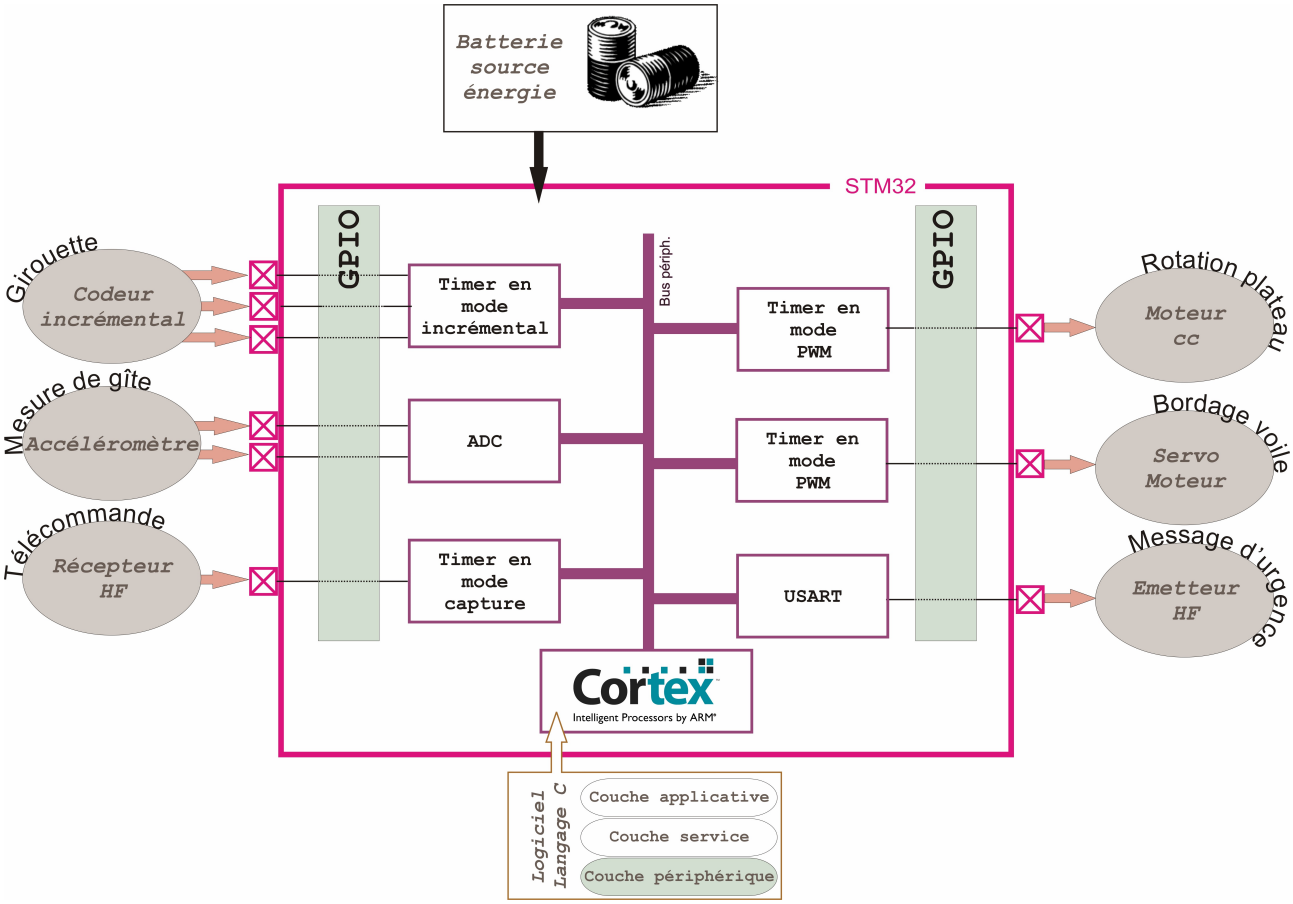
TP *Timer (General Purpose Timer, TIMx)*



1- Les Timers dans le système *Bordage de voile automatique d'un voilier*
2- Le périphérique *Timer*, en général
3- Travail de l'étudiant – fonctionnement des Timers du STM32
4- Annexe : Le circuit d'horloge dans le STM32, introduction

1 Les Timers dans le système *Bordage de voile automatique d'un voilier*



Ces unités *Timer*, déclinées dans plusieurs modes de fonctionnement, vont trouver une large place dans cette application :

- ***Détermination de l'angle du vent par rapport à la proue du bateau*** (girouette), nécessaire pour définir l'allure du bateau (navigation au près, vent arrière, grand largue...). Ceci s'obtient grâce à un capteur de type codeur incrémental associé à un Timer.
- ***Décodage de l'information émise par la télécommande***. Il repose sur la mesure de la durée d'une impulsion. Un Timer est nécessaire.

- ***Rotation du plateau sur lequel repose le voilier***. Celle-ci est rendue possible par un moteur à courant continu. Il est commandé en PWM (*Pulse Width Modulation*). Cette modulation se fait grâce à un Timer.
- ***Commande de bordage de la voile***, basée sur l'envoie périodique d'impulsions calibrées (PWM aussi). Là encore on fera appel à un Timer.
- ***L'ordonnancement*** des activités que vous allez réaliser sera cadencé à intervalles de temps réguliers. Encore un Timer...

Comme nous pouvons le voir, ce périphérique est non seulement très utile, mais en plus il se décline sous maintes fonctions différentes.

Dans un soucis pédagogique, le présent document traitera du Timer dans sa plus simple expression. Le TP suivant fera la lumière sur les interruptions qui lui sont associées. Par la suite, nous approfondirons la fonction PWM du Timer. Pour le reste, vous le découvrirez en complète autonomie lors du projet.

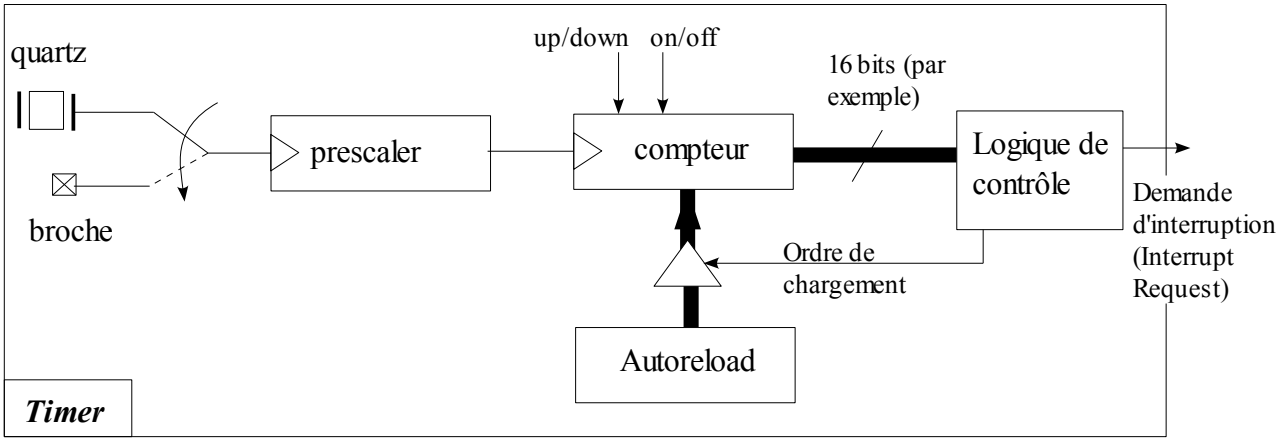
2 Le périphérique *Timer*, en général

Le coeur d'un Timer est un ***compteur électronique***. Celui-ci peut avoir une résolution de 8 bits (0 à 255), 16 bits (0 à 65 535) ou encore 32 bits (0 à 4 294 967 295 !).

Ce Timer possède donc bien évidemment une ***horloge***. Selon la nature de l'horloge, on parlera d'un *fonctionnement en Timer* (l'horloge est dérivée d'un quartz de référence, fixe) ou d'un *fonctionnement en compteur* (l'horloge est dérivée d'une broche du microcontrôleur). L'entrée d'horloge est très souvent précédée d'un ***Prescaler***. Son rôle est d'opérer une première division de la fréquence de l'horloge avant d'attaquer concrètement l'horloge du compteur (voir schéma).

Enfin, précisions que le compteur est très souvent associé à un registre dit ***Autoreload***. Celui-ci contient la valeur de redémarrage du compteur après un débordement (haut ou bas).

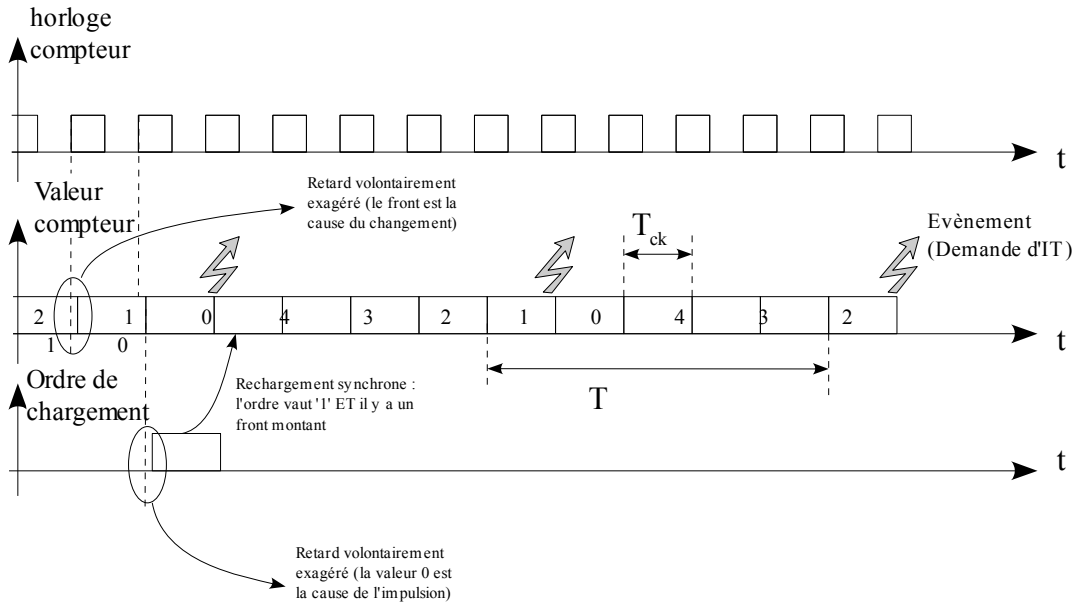
Voici le schéma typique d'un timer dans sa plus simple expression :



Remarque : *La figure précédente est très simpliste, et ne permet d'appréhender le Timer uniquement dans le but de générer un évènement périodique.*

La plupart du temps, le compteur possède une entrée de contrôle *up / down* et le compteur fonctionne de manière purement **synchrone**. Ci-après un chronogramme qui montre le timer en mode décomptage ainsi qu'une analyse de *l'autorelaod*.

On suppose un mode de décomptage (down) et une valeur d'autoreload de 4 et qu'au départ, le compteur a la valeur 2.



Sur cet exemple, la durée T est la période d'évolution de la séquence du Timer. Elle vaut 5 T_{ck}. T_{ck} étant la période d'horloge.

Ainsi, en généralisant, si l'on souhaite une période pour la séquence du Timer de n.T_{ck}, on placera dans le registre T_{ck} la valeur n-1.

La génération de l'évènement (demande d'interruption par exemple, mais pas forcément) se traduit physiquement par une impulsion qui a lieu à l'underflow (décomptage) ou overflow (comptage). L'underflow est le passage de 0 à la valeur suivante (0xFFFF si l'autorelaod n'est pas configuré, valeur de l'autorelaod dans le cas contraire).

3 Travail de l'étudiant – fonctionnement des Timers du STM32

NB: Sur la documentation, un schéma fonctionnel est donné. Il montre des entrées et sorties. **Ce ne sont pas les IO**. Ce sont les entrées et sorties du périphériques qui sont susceptibles d'être reliées aux broches (IO) de la puce (en configurant l'IO sur *alternate function*).

NB: Les timers 2, 3 et 4 sont strictement identiques. Le *timer 1* est amélioré. Mais il est régi par le même jeu de registres, de type *TIM_TypeDef*. Ainsi, bien que certains registres du *timer 1* soient plus fournis en terme de bits de configuration, les fonctions de configurations générales sont les mêmes. C'est pourquoi, nous décidons de créer un module pilote qui regroupe les 4 timers ensemble.

A vous de lire la documentation pour en extraire les registres et bits permettant de configurer un Timer.

Vous remarquerez qu'un Timer est cadencé à l'aide d'une horloge CLK_INT, l'un des problèmes qui se pose est de connaître et de configurer cette horloge.

- Quelles sont les différentes sources d'horloge du STM32 ?
- Quels sont les registres permettant de configurer l'horloge des Timers depuis une source ?

4 Travail de l'étudiant – Keil

4.1 Importation du pilote clock

Un pilote de configuration des horloges a déjà été développé.

⌦ Importez l'archive **clock.zip** et déposer le répertoire dépaqueté *clock* dans *./pilotes*.

⌦ Importez l'archive **projet_vierge.zip** et déposer le répertoire dépaqueté *XXX_projet* dans *./projets* et renommez le en *tp_timer*.

⌦ Modifiez le nouveau projet pour utiliser le pilote **clock** afin de configurer les horloges en utilisant la configuration prédéfinie dans le template.

4.1 Votre premier Timer

⌦ Configurez directement dans le *main()* de votre projet le Timer 2 avec une période de 8 ms.

⌦ En scrutant le flag d'interruption du Timer 2, faite clignoter une diode à une fréquence de 125 Hz et vérifiez en réel son fonctionnement.

⌦ Faites la même chose, mais avec une fréquence de 2 Hz.

4.1 Création du pilote des Timers 1, 2, 3 et 4

⌦ Créez dans le répertoire *./pilotes* un nouveau répertoire *timer*. Ajoutez y un fichier **Timer_1234.c** ainsi que l'entête associée, **Timer_1234.h**.

⌦ Implémentez une fonction qui permet de configurer automatiquement les Timers avec le prototype est le suivant :

```
/**
* Configure les Timers 1, 2, 3 et 4
* @param Timer Pointeur vers le jeu de registres (de type TIM_TypeDef ) du
* timer considéré
* @param Duree_us Intervalle de temps exprimé en µs entre
* deux débordements successifs
* @return Le durée véritable qui a été configurée
**/
            float Timer_1234_Init(TIM_TypeDef *Timer, float Duree_us )
```

NB : Inspirez vous de ce qu'il y a dans le pilote **clock** pour connaître la fréquence d'horloge de votre Timer.

⌦ Vérifiez que la fonction répond bien aux attentes en la testant sur votre application pour faire clignoter une diode.

Annexe : Activation du circuit d'horloge dans le STM32

Lors du précédent TP sur les GPIO, vous avez dû noter une ligne particulière dans le code source GPIO.c :

```
// Activer horloges A, B et C
(RCC->APB2ENR)=(RCC->APB2ENR) | RCC_IOPAEN | RCC_IOPBEN | RCC_IOPCEN;
```

Ceci est une particularité du micro-contrôleur STM32. Chaque périphérique n'est pas, par défaut, relié à une horloge. L'intérêt est **l'économie d'énergie**. En effet, un circuit électronique relié à une horloge, même s'il ne pilote pas de dispositif, dissipe une puissance au cube de la fréquence d'horloge.

C'est la raison pour laquelle, il faut, pour chaque périphérique activer l'horloge locale au périphérique (repérer le bit de validation dans le jeu de registres RCC).